Generalizing CNNs for data structured on locations irregularly spaced out

Jean-Charles Vialatte[1,2], Vincent Gripon[2], Grégoire Mercier[2]

[1]Cityzen Data
Guipavas, Brittany, France

[2]Telecom Bretagne
Brest, Brittany, France

[1]`jean-charles.vialatte@cityzendata.com`
[2]`{jc.vialatte, vincent.gripon, gregoire.mercier}@telecom-bretagne.eu`

Abstract

Convolutional Neural Networks (CNNs) have become the state-of-the-art in supervised learning vision tasks. Their convolutional filters are of paramount importance for they allow to learn patterns while disregarding their locations in input images. When facing highly irregular domains, generalized convolutional operators based on an underlying graph structure have been proposed. However, these operators do not exactly match standard ones on grid graphs, and introduce unwanted additional invariance (e.g. with regards to rotations). We propose a novel approach to generalize CNNs to irregular domains using weight sharing and graph-based operators. Using experiments, we show that these models resemble CNNs on regular domains and offer better performance than multilayer perceptrons on distorted ones.

1 Introduction

CNNs[1] are state-of-the-art for performing supervised learning on data defined on lattices [2]. Contrary to classical multilayer perceptrons (MLPs) [3], CNNs take advantage of the underlying structure of the inputs. When facing data defined on irregular domains, CNNs cannot always be directly applied even if the data may present an exploitable underlying structure.

An example of such data are spatio-temporal time series generated by a set of Internet of Things devices. They typically consist of datapoints irregularly spaced out on a Euclidean space. As a result, a graph G can be defined where the vertices are the devices and the edges connect neighbouring ones. Other examples include signals on graphs including brain imaging, transport networks, bag of words graphs... In all these examples, a signal can generally be seen as a vector in $\mathbb{R}^d$ where d is the order of the graph. As such, each vertex is associated with a coordinate, and the edges weights represent some association between the corresponding vertices.

Disregarding the graph G, MLPs can be applied on these datasets. We are interested in defining convolutional operators that are able to exploit G as well as a possible embedding Euclidean space. Our motivation is to imitate the gain of performance allowed by CNNs over MLPs on irregular domains.

In graph signal processing, extended convolutional operators using spectral graph theory [4] have been proposed [5]. These operators have been applied to deep learning [6] and obtain performance similar to CNNs on regular domains, despite the fact they differ from classical convolutions. However, for slightly distorted domains, the obtained operators become non-localized, thus failing taking account of the underlying structure.

We propose another approach that generalizes convolutional operators by taking into account G. Namely, we make sure that the proposed solution has properties inherent to convolutions: linearity, locality and kernel weight sharing. We then apply it directly to the graph structure. We use it as a substitution of a convolutional layer in a CNN and stress our resulting technique on comparative benchmarks, showing a significant improvement compared with a MLP. The obtained operator happens to exactly be the classical convolutional one when applied on regular domains.

The outline of the paper is as follows. In Section 2, we discuss related work. In Section 3 we introduce our proposed operator. In Section 4 we explain how to apply it to CNN-like structures. Section 5 contains the experiments. Section 6 is a conclusion.

2 Related Works

For graph-structured data, Bruna et al. [6] have proposed an extension of the CNN using graph signal processing theory [5]. Their convolution is defined in the spectral domain related to the Laplacian matrix of the graph. As such, in the case where the graph is a lattice, the construction is analogous to the regular convolution defined in the Fourier domain. The operation is defined as spectral multipliers obtained by smooth interpolation of a weight kernel, and they explain how it ensures the localization property. In that paper, they also define a construction that creates a multi-resolution structure of a graph, for allowing it to support a deep learning architecture. Henaff et al. [7] have extended Bruna's spectral network to large scale classification tasks and have proposed both supervised and unsupervided methods to find an underlying graph structure when it isn't already given.

However, when the graph is irregular, they partly lose the localization property of their convolution is partially lost. As the spectral domain is undirected with respect to the graph domain, some sort of rotation invariances are also introduced. Hence, the results in the graph domain aren't ressembling to those of a convolution. In our case, we want to define a convolution supported locally. Moreover, we want that every input can be defined on a different underlying graph structure, so that the learnt filters can be applied on data embedded in the same space regardless of what structure have supported the convolution during the training.

These properties are also retained by the convolution defined in the ShapeNet paper[8], which defines a convolution for data living on non-euclidean manifolds. I.e their construction does maintain the locality and allow for reusing the learnt filters on other manifolds. However, it requires at least a manifold embedding of the data and a geodesic polar coordinates system. Although being less specific, our proposed method will be a strict generalization of CNN in the sense that CNNs are a special case of it.

In the case where the data is sparse, Graham [9] has proposed a framework to implement a spatially sparse CNN efficiently. If the underlying graph structure is embedded into an Euclidean space, the convolution we propose in this paper can be seen as spatially sparse too, in the sense that the data has only non-zero coordinates on the vertices of the graph it is defined on. In our case we want to define a convolution for which the inputs can have values on any point of the embedding space, whereas in the regular case inputs can only have values on vertices of an underlying grid.

3 Proposed convolution operator

3.1 Definitions

Let S be a set of points, such that each one defines a set of neighbourhoods.

An *entry* e of a *dataset* $\mathcal{D}$ is a column vector that can be of any size, for which each dimension represents a *value taken by* e at a certain point $u \in S$. u is said to be *activated* by e. A point u can be associated to at most one dimension of e. If it is the ith dimension of e, then we denote the value taken by e at u by either e_u or e_i. $\mathcal{D}$ is said to be *embedded* in S.

We say that two entries e and e' are *homogeneous* if they have the same size and if their dimensions are always associated to the same points.

3.2 Formal description of the *generalized* convolution

Let's denote by $\mathcal{C}$ a generalized convolution operator. We want it to observe the following conditions:

- Linearity
- Locality
- Kernel weight sharing

As $\mathcal{C}$ must be linear, then for any entry e, there is a matrix W^e such that $\mathcal{C}(e) = W^e e$. Unless the entries are all homogeneous, W^e depends on e. For example, in the case of regular convolutions on images, W is a Toeplitz matrix and doesn't depend on e.

In order to meet the locality condition, we first want that the coordinates of $\mathcal{C}(e)$ have a local meaning. To this end, we impose that $\mathcal{C}(e)$ lives in the same space as e, and that $\mathcal{C}(e)$ and e are homogeneous. Secondly, for each u, we want $\mathcal{C}(e)_u$ to be only function of values taken by e at points contained in a certain neighbourhood of u. It results that rows of W^e are generally sparse.

Let's attribute to $\mathcal{C}$ a kernel of n weights in the form of a row vector $w = (w_1, w_2, .., w_n)$, and let's define the set of allocation matrices $\mathcal{A}$ as the set of binary matrices that have at most one non-zero coordinate per column. As $\mathcal{C}$ must share its weights across the activated points, then for each row W_i^e of W^e, there is an allocation matrix $A_i^e \in \mathcal{A}$ such that $W_i^e = wA_i^e$. To maintain locality, the jth column of A_i^e must have a non-zero coordinate if and only if the ith and jth activated points are in a same neighbourhood.

Let's A^e denotes the block column vector that has the matrices A_i^e for attributes, and let's $\otimes$ denotes the tensor product. Hence, $\mathcal{C}$ is defined by a weight kernel w and an allocation map $e \mapsto A^e$ that maintains locality, such that

$$\mathcal{C}(e) = (w \otimes A^e) \cdot e \tag{1}$$

3.3 The underlying graph that supports the generalized convolution

As vertices, the set of activated points of an entry e defines a complete oriented graph that we call G^e. If all the entries are homogeneous, then G^e is said to be *static*. In this case, we note G. Otherwise, there can be one different graph per entry so the kernel of weights can be re-used for new entries defined on different graphs.

Suppose we are given $u \mapsto \mathcal{V}_u$ which maps each $u \in S$ to a neighbourhood $\mathcal{V}_u$. We then define $G_{\mathcal{V}}^e$ as the subgraph of G^e such that it contains the edge (u', u) if and only if $u' \in \mathcal{V}_u$. Let $a_{\mathcal{V}} : e \mapsto A^e$ be an allocation map such that the jth column of A_i^e have a non-zero coordinate if and only if (e_i, e_j) is an edge of $G_{\mathcal{V}}^e$.

Then the generalized convolution of e by the couple $(w, a_{\mathcal{V}})$ is supported by the underlying graph $G_{\mathcal{V}}^e$ in the sense that $W_{\mathcal{V}}^e = w \otimes a_{\mathcal{V}}(e)$ is its adjacency matrix. Note that the underlying graph of a regular convolution is a lattice.

Also note that the family $(\mathcal{V}_u)_{u \in S}$ can be seen as local receptive fields for the generalized convolution, and that the map $a_{\mathcal{V}}$ can be seen as if it were distributing the kernel weights into each $\mathcal{V}_u$.

Remarks

The generalized convolution has been defined here as an operation between a kernel weight and a vector. If $G = (E, V)$ denotes a graph, then such operator $*$ with respect to a third rank tensor A could also have been defined as a bilinear operator between two graph signals $f, g \in \mathbb{R}^E$:

$$f * g = (f^\top \otimes A) \cdot g \tag{2}$$

$$\forall v \in V, \ (f * g)(v) = f^\top A_v \ g, \ with \ A_v \in \mathcal{A} \tag{3}$$

Note that the underlying graph depends on the entry. So the learnt filter w is reusable regardless of the entry's underlying graph structure. This is not the case for convolutions defined on fixed graphs like in [6, 7].

3.4 Example of a generalized convolution shaped by a rectangular window

Let $\mathbb{E}$ be a two-dimensional Euclidean space and let's suppose here that $S = \mathbb{E}$.

Let's denote by $\mathcal{C}_\mathcal{R}$, a generalized convolution shaped by a rectangular window $\mathcal{R}$. We suppose that its weight kernel w is of size $N_w = (2p+1)(2q+1)$ and that $\mathcal{R}$ is of width $(2p+1)\mu$ and height $(2q+1)\mu$, where μ is a given unit scale.

Let $G^e_{p,q}$ be the subgraph of G^e such that it contains the edge (u', u) if and only if $|u'_x - u_x| \leq (p+\frac{1}{2})\mu$ and $|u'_y - u_y| \leq (q+\frac{1}{2})\mu$. In other terms, $G^e_{p,q}$ connects a vertex u to every vertex contained in $\mathcal{R}$ when centered on u.

Then, we define $\mathcal{C}_\mathcal{R}$ as being supported by $G^e_{p,q}$. As such, its adjacency matrix acts as the convolution operator. At this point, we still need to affect the kernel weights to its non-zero coordinates, via the edges of $G^e_{p,q}$. This amounts for defining explicitely the map $e \mapsto A^e$ for $\mathcal{C}_\mathcal{R}$. To this end, let's consider the grid of same size as that rectangle, which breaks it down into $(2p+1)(2q+1)$ squares of side length μ, and let's associate a different weight to each square. Then, for each edge (u', u), we affect to it the weight associated with the square within which u' falls when the grid is centered on u. This procedure allows for the weights to be shared across the edges. It is illustrated on figure 1.

Note that if the entries are homogeneous and the activated points are vertices of a regular grid, then the matrix W, independant of e, is a Toepliz matrix which acts as a regular convolution operator on the entries e. In this case, $\mathcal{C}_\mathcal{R}$ is just a regular convolution. For example, this is the case in section 3.5.
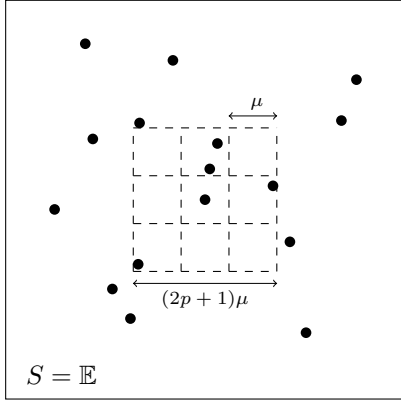

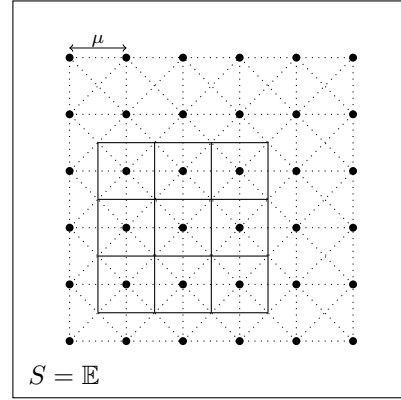Figure 1: Example of a moving grid. The grid defines the allocation of the kernel weights.

Figure 2: On regular domains, the moving grid allocates the weights similarly to the moving window of a standard convolution.

3.5 Link with the standard convolution on image datasets

Let $\mathcal{D}$ be an image dataset. Entries of $\mathcal{D}$ are homogeneous and their dimensions represent the value at each pixel. In this case, we can set $S = \mathbb{E}$, of dimension 2, such that each pixel is located at entire coordinates. More precisely, if the images are of width n and height m, then the pixels are located at coordinates $(i, j) \in \{0, 1, 2, ..., n\} \times \{0, 1, 2, ..., m\}$. Hence, the pixels lie on a regular grid and thus are spaced out by a constant distance $\mu = 1$.

Let's consider the static underlying graph $G_{p,q}$ and the generalized convolution by a rectangular window $\mathcal{C}_\mathcal{R}$, as defined in the former section. Then, applying the same weight allocation strategy will lead to affect every weight of the kernel into the moving window $\mathcal{R}$. Except on the border, one and only one pixel will fall into each square of the moving grid at each position, as depicted in figure 2. Indeed, $\mathcal{R}$ behaves exactly like a moving window of the standard convolution, except that it considers that the images are padded with zeroes on the borders.

4 Application to CNNs

4.1 Neural network interpretation

Let $\mathcal{L}^d$ and $\mathcal{L}^{d+1}$ be two layers of neurons, such that forward-propagation is defined from $\mathcal{L}^d$ to $\mathcal{L}^{d+1}$. Let's define such layers as a set of neurons being located in S. These layers must contain as many neurons as points that can be activated. In other terms, $S \cong \mathcal{L}^d \cong \mathcal{L}^{d+1}$. As such, we will abusively use the term of *neuron* instead of *point*.
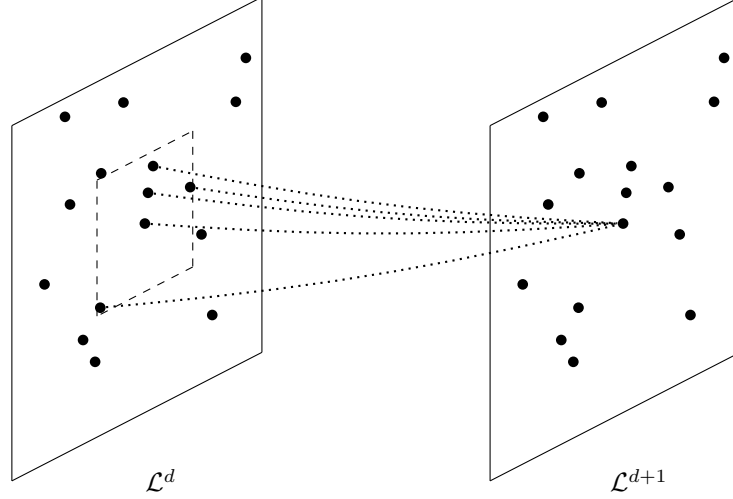

Figure 3: Generalized convolution between two layers of neurons.

The *generalized* convolution between these two layers can be interpreted as follow. An entry e activates the same N neurons in each layer. Then, a convolution shape takes N positions onto $\mathcal{L}^d$, each position being associated to one of the activated neurons of $\mathcal{L}^{d+1}$. At each position, connections are drawn from the activated neurons located inside the convolution shape in destination to the associated neuron. And a subset of weights from w are affected to these connections, according to a weight sharing strategy defined by an allocation map. Figure 3 illustrates a convolution shaped by a rectangular window.

The forward and backward propagation between $\mathcal{L}^d$ and $\mathcal{L}^{d+1}$ are applied using the described neurons and connections. After a generalized convolution operation, an activation function is applied on the output neurons. Then a pooling operation is done spatially: the input layer is divided into patches of same size, and all activated neurons included in this patch are pooled together. Unlike a standard pooling operation, the number of activated neurons in a patch may vary.

Generalized convolution layers can be vectorized. They can have multiple input channels and multiple feature maps. They shall naturally be placed into the same kind of deep neural network structure than in a CNN. Thus, they are for the irregular input spaces what the standard convolution layers are for regular input spaces.

4.2 Implementation

There are two main strategies to implement the propagations. The first one is to start from (1), derive it and vectorize it. It implies handling semi-sparse representations to minimize memory consumption and to use adequate semi-sparse tensor products.

Instead, we decide to use the neural network interpretation and the underlying graph structure whose edges amount for neurons connections. By this mean, the sparse part of the computations is included via the use of this graph. Also, computations on each edge can be parallelized.

5

5 Experiments

In order to measure the gain of performance allowed by the generalized CNN over MLP on irregular domains, we made a series of benchmarks on distorted versions of the MNIST dataset [10], consisting of images of 28x28 pixels. To distort the input domain, we plunged the images into a 2-d euclidean space by giving entire coordinates to pixels. Then, we applied a gaussian displacement on each pixel, thus making the data irregular and unsuitable for regular convolutions. For multiple values of standard deviation of the displacement, we trained a generalized CNN and compared it with a MLP that has the same number of parameters. We choosed a simple yet standard architecture in order to better see the impact of generalized layers.

The architecture used is the following: a generalized convolution layer with relu [11] and max pooling, made of 20 feature maps, followed by a dense layer and a softmax output layer. The generalized convolution is shaped by a rectangular window of width and height 5μ where the unit scale μ is chosen to be equal to original distance between two pixels. The max-pooling is done with square patches of side length 2μ. The dense layer is composed of 500 hidden units and is terminated by relu activation as well. In order to have the same number of parameters, the compared MLP have 2 dense layers of 500 hidden units each and is followed by the same output layer. For training, we used stochastic gradient descent [12] with Nesterov momentum [13] and a bit of L2 regularization [14].
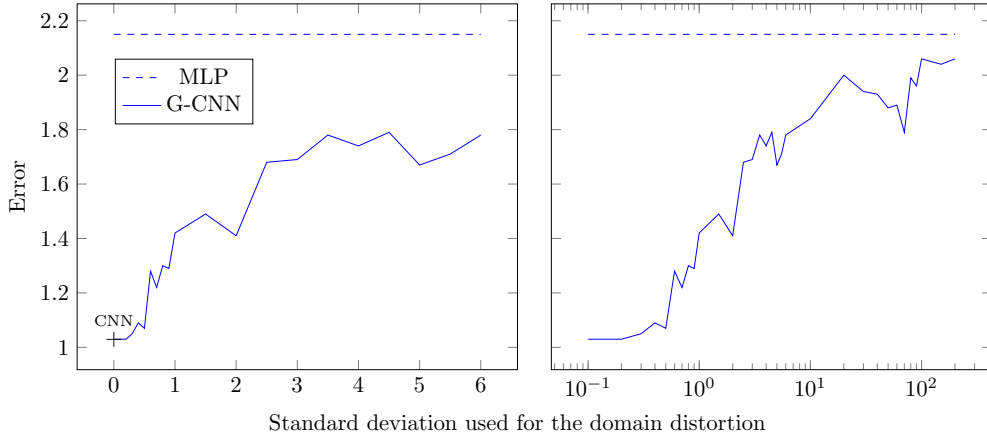

Figure 4: Results for Generalized CNN and MLP

The plot drawn on figure 4 illustrates the gain of performance of a generalized convolutional layer over a dense layer with equal number of parameters. After 15 epochs for both, it shows that the generalized CNN on a distorted domain performs better than the MLP. Indeed, in the case of no domain distortion, the score is the same than a CNN with zero-padding. The error rate goes up a bit with distortion. But even at 200μ, the generalized CNN still performs better than the MLP.

6 Conclusion and future work

In this paper, we have defined a generalized convolution operator. This operator makes possible to transport the CNN paradigm to irregular domains. It retains the proprieties of a regular convolutional operator. Namely, it is linear, supported locally and uses the same kernel of weights for each local operation. The generalized convolution operator can then naturally be used instead of convolutional layers in a deep learning framework. Typically, the created model is well suited for input data that has an underlying graph structure.

The definition of this operator is flexible enough for it allows to adapt its weight-allocation map to any input domain, so that depending on the case, the distribution of the kernel weight can be done in a way that is natural for this domain. However, in some cases, there is no natural way but multiple acceptable methods to define the weight allocation. In further works, we plan to study these methods. We also plan to apply the generalized operator on unsupervised learning tasks.

6

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[3] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[4] F. R. K. Chung, *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92)*. American Mathematical Society, 1996.

[5] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, no. EPFL-ARTICLE-189192, pp. 83–98, 2013.

[6] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[7] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[8] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, "Shapenet: Convolutional neural networks on non-euclidean manifolds," tech. rep., 2015.

[9] B. Graham, "Spatially-sparse convolutional neural networks," *arXiv preprint arXiv:1409.6070*, 2014.

[10] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits," 1998.

[11] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

[12] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[13] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th international conference on machine learning (ICML-13)*, pp. 1139–1147, 2013.

[14] A. Y. Ng, "Feature selection, l 1 vs. l 2 regularization, and rotational invariance," in *Proceedings of the twenty-first international conference on Machine learning*, p. 78, ACM, 2004.